ASSIGNMENT-03

Y.Sahasra

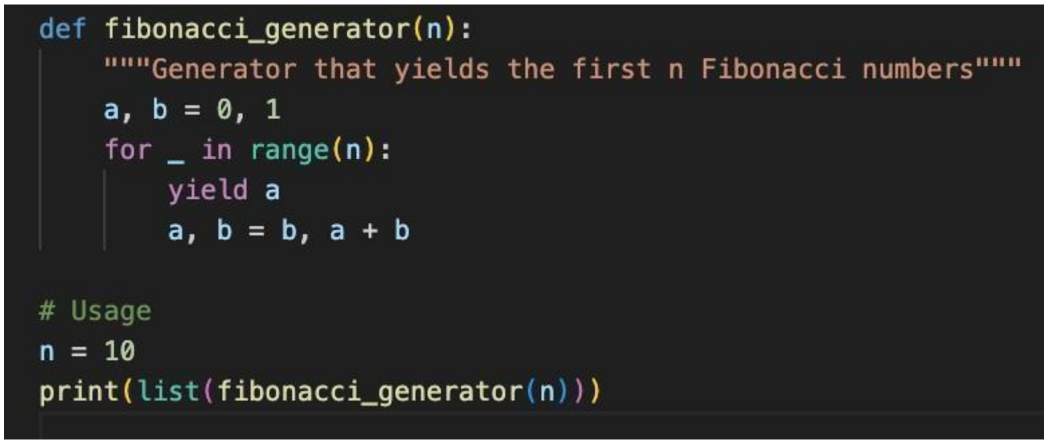
2303A51269

Batch: 19

Task 1: Zero-shot Prompt – Fibonacci Series Generator

Scenario : In this task, a zero-shot prompting technique was used. A single comment prompt was written without providing any examples, instructing GitHub Copilot to generate a Python function that prints the first N Fibonacci numbers.

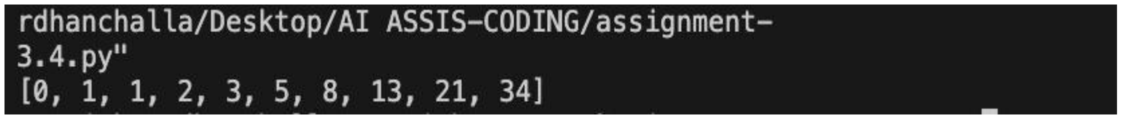
Prompt: # Write a Python function to print the first N Fibonacci numbers Code:



```python
def fibonacci_generator(n):
    """Generator that yields the first n Fibonacci numbers"""
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a + b

# Usage
n = 10
print(list(fibonacci_generator(n)))
```

Result:



```
rdhanchalla/Desktop/AI ASSIS-CODING/assignment-
3.4.py"
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

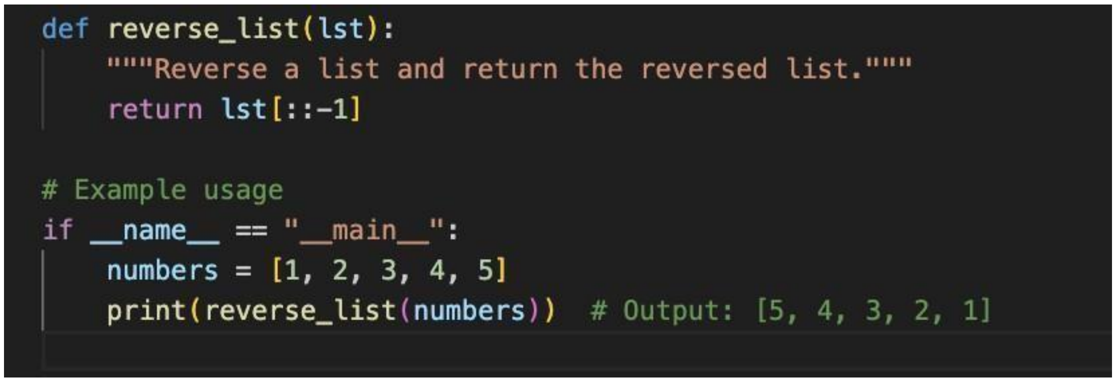
Observation:

The zero-shot prompt was sufficient for Copilot to correctly infer the Fibonacci logic, even without any examples or additional context. However, the function behavior depended heavily on Copilot's prior training, and the output format was assumed rather than explicitly defined. This shows that zero-shot prompting works well for well-known problems but may lack consistency for ambiguous or complex tasks.

Task 2: One-shot Prompt – List Reversal Function. In this task, a one-shot prompting approach was used by providing a single example along with the instruction to help Copilot generate a correct list reversal function

Prompt: # Write a Python function to reverse a list #

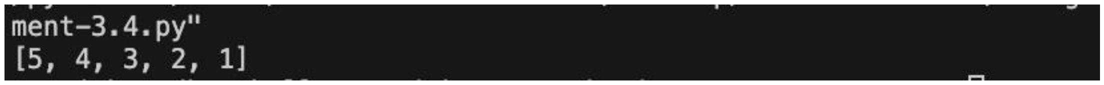
Example: input [1, 2, 3] -> output [3, 2, 1] Code:



```python
def reverse_list(lst):
    """Reverse a list and return the reversed list."""
    return lst[::-1]

# Example usage
if __name__ == "__main__":
    numbers = [1, 2, 3, 4, 5]
    print(reverse_list(numbers))  # Output: [5, 4, 3, 2, 1]
```

Result:



```
ment-3.4.py"
[5, 4, 3, 2, 1]
```

Observation:
Providing one example significantly improved Copilot's accuracy and confidence in choosing an optimal approach. The generated solution was concise and efficient, using Python slicing. Compared to zero-shot prompting, one-shot prompting reduced ambiguity and guided Copilot toward the expected output format and logic

Task 3: Few-shot Prompt – String Pattern Matching

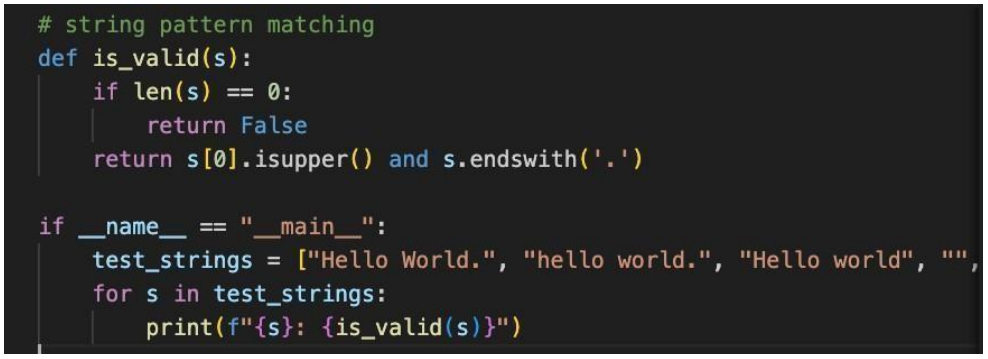
Scenario: This task used a few-shot prompting technique by providing multiple examples to help Copilot understand a specific string validation pattern.

Prompt: # Write a function to check if a string starts with a capital letter and ends with a period

# Example: "Hello." -> True

# Example: "hello." -> False

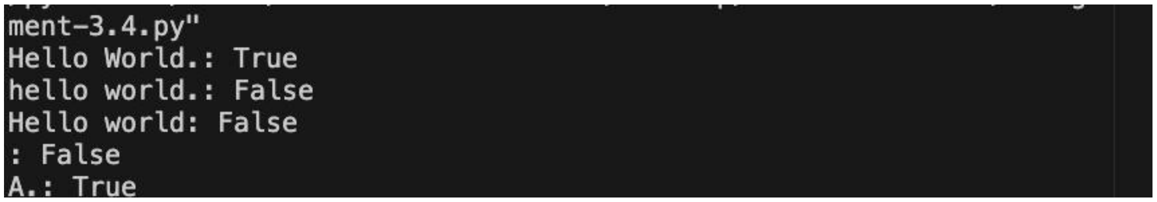
# Example: "Hello" -> False Code:



```python
# string pattern matching
def is_valid(s):
    if len(s) == 0:
        return False
    return s[0].isupper() and s.endswith('.')

if __name__ == "__main__":
    test_strings = ["Hello World.", "hello world.", "Hello world", "",
    for s in test_strings:
        print(f"{s}: {is_valid(s)}")
```

Result:



```
ment-3.4.py"
Hello World.: True
hello world.: False
Hello world: False
: False
A.: True
```

Observation:
The few-shot prompt enabled Copilot to accurately identify the string pattern requirements and generate a precise validation function. The multiple examples clarified edge cases and reduced misinterpretation. This demonstrates that few-shot prompting is highly effective when pattern recognition or conditional logic is involved.

Task 4: Zero-shot vs Few-shot – Email Validator

You are participating in a code review session. This task compares zero-shot and few-shot prompting by generating two versions of an email validation function and analyzing their differences

Prompt: Zero-Shot Prompt: # Write a Python function to validate an email address

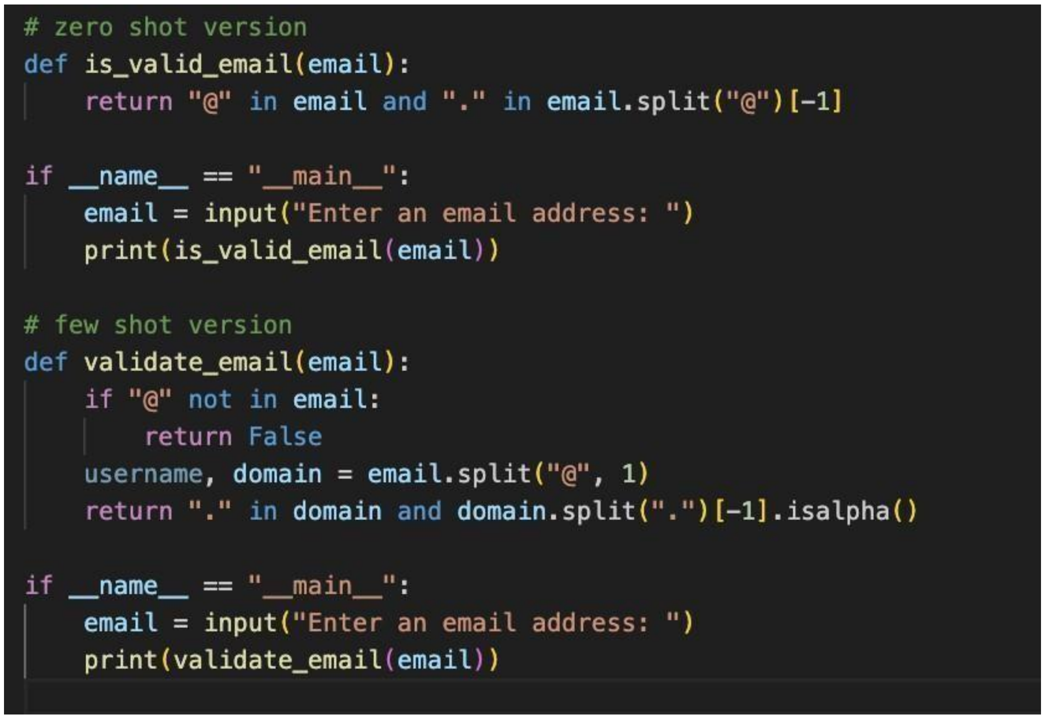
Prompt: Few-Shot Prompt: # Write a Python function to validate an email address

# Example: "test@gmail.com" -> True

# Example: "testgmail.com" -> False

# Example: "test@com" -> False

Code:



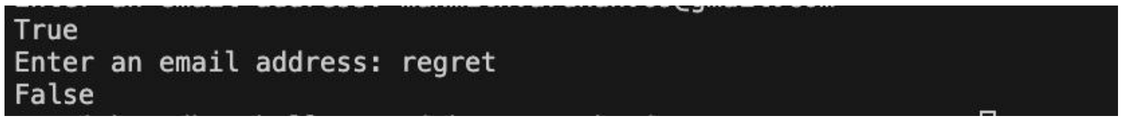
```python
# zero shot version
def is_valid_email(email):
    return "@" in email and "." in email.split("@")[-1]

if __name__ == "__main__":
    email = input("Enter an email address: ")
    print(is_valid_email(email))

# few shot version
def validate_email(email):
    if "@" not in email:
        return False
    username, domain = email.split("@", 1)
    return "." in domain and domain.split(".")[-1].isalpha()

if __name__ == "__main__":
    email = input("Enter an email address: ")
    print(validate_email(email))
```

Result:



```
True
Enter an email address: regret
False
```

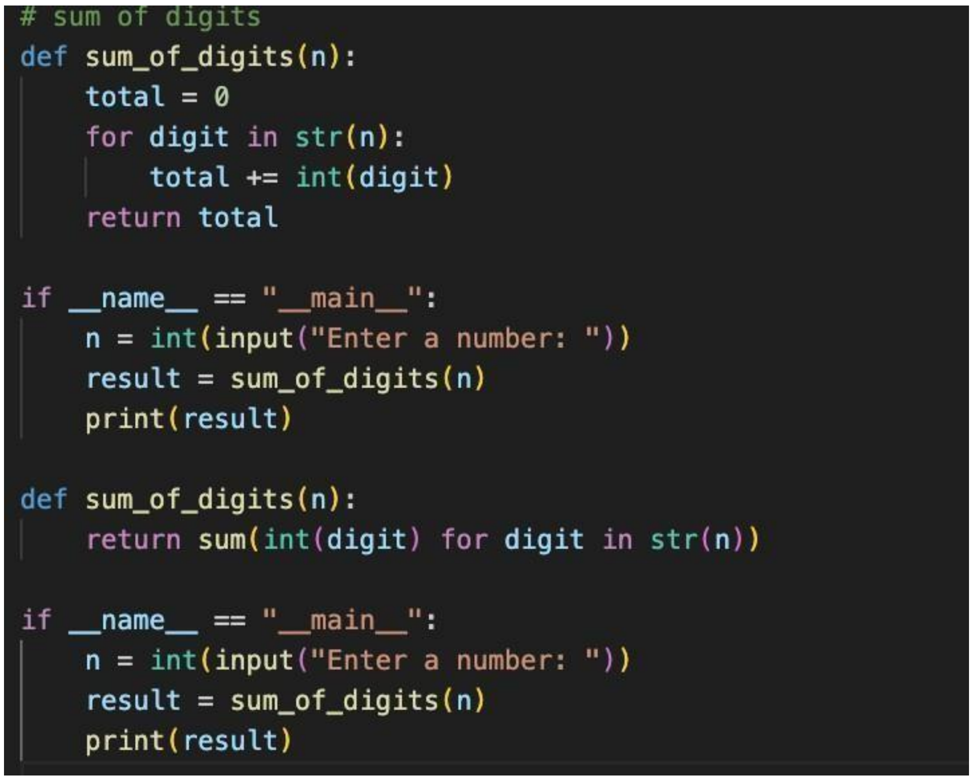
Observation

The zero-shot version produced a very basic and unreliable validation logic, while the few-shot prompt resulted in a more structured and realistic solution. The examples guided Copilot to include domain checks and input validation, significantly improving reliability. This comparison clearly highlights the advantage of few-shot prompting for real-world validation tasks.

Task 5: Prompt Tuning – Summing Digits of a Number. In this task, two different prompt styles were used to study how prompt tuning affects code quality and optimization.

Prompt: Style-1:Generic Prompt # Write a function to return the sum of digits of a number Prompt with I/O Example: # Write a function to return the sum of digits of a number # Example: sum_of_digits(123) -> 6

Code:



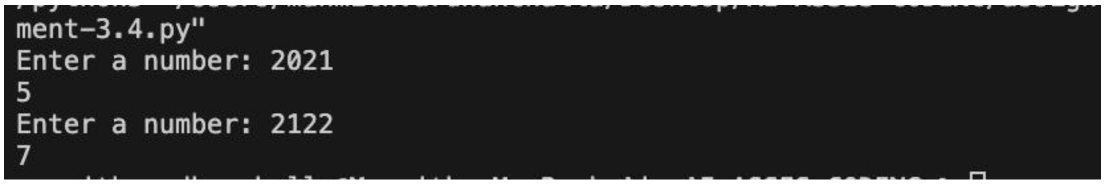
```python
# sum of digits
def sum_of_digits(n):
    total = 0
    for digit in str(n):
        total += int(digit)
    return total

if __name__ == "__main__":
    n = int(input("Enter a number: "))
    result = sum_of_digits(n)
    print(result)

def sum_of_digits(n):
    return sum(int(digit) for digit in str(n))

if __name__ == "__main__":
    n = int(input("Enter a number: "))
    result = sum_of_digits(n)
    print(result)
```

Result:



```
ment-3.4.py"
Enter a number: 2021
5
Enter a number: 2122
7
```

Observation:

The prompt that included an input-output example produced a cleaner and more optimized implementation. The example encouraged Copilot to generate concise and Pythonic code using built-in functions. This demonstrates that prompt tuning with examples not only improves correctness but also enhances code quality and efficiency.